1.0 Introduction

The Defense Information Infrastructure (DII) Common Operating Environment (COE) originated with a simple observation about command and control systems: certain functions (mapping, track management, communication interfaces, etc.) are so fundamental that they are required for virtually every command and control system. Yet these functions are built over and over again in incompatible ways even when the requirements are the same, or vary only slightly, between systems. If these common functions could be extracted, implemented as a set of extensible low level building blocks, and made readily available to system designers, development schedules could be accelerated and substantial savings could be achieved through software reuse. Moreover, interoperability would be significantly improved because common software is used across systems for common functions.

This observation led to the development of the DII COE which is presently used in two systems: the Global Command and Control System (GCCS), and the Global Combat Support System (GCSS). Both systems use the same infrastructure and integration approach, and the same COE components for functions that are common.

GCCS is a C4I system with two main objectives: the near-term replacement of the World-Wide Military Command and Control System (WWMCCS) and the implementation of the C4I For the Warrior concept. Functionally, as a C4I For the Warrior system, GCCS includes multiple workstations cooperating in a distributed LAN/WAN environment. Key features include "push/pull" data exchange, data processing, sensor fusion, dynamic situation display, analysis and briefing support, and maintenance of a common tactical picture among distributed GCCS sites. GCCS is already fielded at a number of operational CINCs.

GCSS is presently under development and is targeted for the warfighting support functions (logistics, transportation, etc.) to provide a system that is fully interoperable with the warfighter C4I system. Implemented to its fullest potential, GCSS will provide both warfighter support to include reachback from deployed commanders into the CONUS sustaining base infrastructure, and cross functional integration on a single workstation platform.

Initial COE development was driven by the near-term requirement to build a suitable WWMCCS replacement. WWMCCS maintenance costs are significant and the system is rapidly reaching the point of technical obsolescence. A significant component of the COE challenge is to strategically position the system architecture so as to be able to take advantage of technological advances. At the same time, the system must not sacrifice quality, stability, or functionality already in the hands of the warrior. In keeping with current DoD trends, the COE emphasizes use of commercial products and standards where applicable to leverage investments made by commercial industry.

To achieve the near-term WWMCCS replacement objective, technical experts and program managers from each of the services, DODIIS, DMA, and other interested agencies met for several months beginning in the fall of 1993. Participants proposed candidate systems as a possible starting point for the COE architecture, or as a suitable candidate for providing capabilities to meet WWMCCS replacement requirements. None of the candidate systems met all requirements, but it was clear that a combination of the "best" from several systems could produce a near term system that would be suitable for WWMCCS replacement. Moreover, an infrastructure could be put into place and a migration strategy defined to preserve legacy systems until migration to the intended architecture could be realized.

The cornerstone architectural concept jointly developed during these series of meetings is the DII COE. The present COE is composed of software contributed from several candidate systems evaluated by this joint engineering team. It is being expanded to include global data management and workflow management for GCSS logistics applications. It will expand further as more functional areas desire to employ its services in areas such as Electronic Commerce/Electronic Data Interchange (EC/EDI), transportation, base support, personnel, health affairs, and finance. The COE is described more completely in later chapters of this document. This document also describes technical information required to properly access and extend software contained within the COE.

An initial proof-of-concept system, GCCS 1.0, was created and installed in early 1994 at one operational site to validate the approach and to receive early feedback. GCCS 1.1 followed in the summer of 1994 and was the first attempt to integrate software from the Army AWIS and Navy JMCIS programs as initial COE components. GCCS 1.1 included mission applications from a variety of other programs operating in a "federated" mode. That is, constructed so as to be able to run on the same hardware without interfering with other software, but not yet able to effectively share data between applications. This successful effort allowed GCCS 1.1 to be installed and tested at beta sites and was used at certain operational sites to monitor events during the 1994 Haiti crisis. GCCS 2.0 fielding began in early 1995 at a number of operational sites. GCCS 2.1 was fielded in mid-1995. A prototype version of GCCS 2.2 was the basis for JWID 95. The 2.0 series marks the real beginning of the DII COE concept. Its use is crucial in being able to rapidly integrate software from candidate programs to successfully build a baseline with an ever increasing level of functionality.

The DII COE has its roots in command and control, but the principles and implementation described in this document are not unique to GCCS nor GCSS.

The principles and implementation are not limited to command and control or logistics applications, but are readily applicable to many other application areas. The specific software components selected for inclusion in the COE determine the mission areas that the COE can address.

The concepts herein represent the culmination of open systems evolutionary development from both industry and government with contributions from each of the services. The resulting COE architecture is an innovative framework for designing and building military systems. Because it reuses software contributed by service/agency programs, it utilizes field proven software for common C4I functions. The engineering procedures for adding new capabilities and integrating systems are mature, and have been used for several Navy JMCIS releases as well as in all GCCS production releases. The end result is a strategy for fielding systems with increased interoperability, reduced development time, increased operational capability, minimized technical obsolescence, minimal training requirements, and minimized life cycle costs.

1.1 The DII COE Concept

The DII COE concept is a fundamentally new approach that is much broader in scope than simple software reuse. Software reuse itself is not a new idea. Unfortunately, most software reuse approaches to date have been less than satisfactory. Reuse approaches have generally emphasized the development of a large software repository from which designers may pick and choose modules, or elect to rebuild modules from scratch. It is not sufficient to have a large repository, and too much freedom of choice leads to interoperability problems and duplication of effort. This rapidly negates the advantages of software reuse.

The DII COE does emphasize both software reuse and interoperability, but its principles are more far reaching and innovative. The COE concept encompasses:

¥ an architecture and approach for building interoperable systems,
¥ an infrastructure for supporting mission area applications,
¥ a rigorous definition of the runtime execution environment,
¥ a rigorous set of requirements for achieving COE compliance,
¥ an automated toolset for enforcing COE principles and measuring COE compliance,
¥ an automated process for software integration,
¥ a collection of reusable software components,
¥ an approach and methodology for software reuse,
¥ a set of APIs for accessing COE components, and
¥ an electronic process for submitting/retrieving software components to/from the COE software repository.

This document first and foremost describes *how* modules must interact in the target system. System architects and software developers retain considerable freedom in building the system, but runtime environmental conflicts are identified and resolved through automated tools that enforce COE principles. An important side effect is that traditional integration tasks become the responsibility of the developer. Developers are required to integrate and test their software within the COE prior to delivering it to the government. This simplifies integration because it is performed by those who best understand the software design (the original developers), it reduces the cost because integration is performed earlier and at a lower level in the process, and it allows the government to concentrate on validation instead of integration.

In the context of this document, the COE must be understood as a multi-faceted concept. Proper understanding of how the many facets interact is important in appreciating the scope and power of the DII COE, and to avoid confusion in understanding COE material. The next subsection deals with three specific facets in more detail: the COE as a system foundation, the COE as an architecture, and the COE as an implementation strategy.

To view the COE as a C4I system is incorrect because it misses the fundamental point that the COE is *not* a system; it is a *foundation* for building an open system. This viewpoint also makes fielding and update schedules confusing because it fails to account for the impact of the evolutionary development strategy. To view the COE as GCCS or just an architecture gives the mistaken impression that its principles are limited to the GCCS program. GCCS is simply the first system build on top of the DII COE while GCSS is in progress. This view also fails to account for the fact that a baseline already exists composed of components selected from mature service/agency programs. Finally, to view the COE as just an implementation strategy is a limited perspective because it fails to account for the fact that there is a near term real world objective (WWMCCS replacement). It ignores the evolutionary nature of the COE and mission applications development, and it ignores the implied requirement to provide an easy update mechanism for operational sites.

1.1.1 The DII COE As A System Foundation

Figure 1-1 shows how the DII COE serves as a foundation for building multiple systems. The shaded box shows two types of reusable software: the operating system and COE components. Chapter 2 describes the COE components in more detail, and the supported operating systems. For the present discussion, it is sufficient to note that these components are accessed through APIs and that they form the architectural backbone of the target system.

Building a target system, such as GCCS or GCSS, is largely a matter of combining COE components with mission specific software. The COE infrastructure manages the flow of data through the system, both internally and externally. Mission specific software is mostly concerned with requesting data from the COE and then presenting it in a form that is most meaningful to the operator (e.g., as a pie chart, in tabular form, as a graph). The COE provides the necessary primitives for such data manipulation, and has the necessary information about where the requested data is stored, whether locally or remotely across the LAN/WAN. This frees the system designer to concentrate on meaningful data presentation and not on the mechanics of data manipulation, network communications, database storage, etc.

It must be kept in mind, however, that there is only one COE. Each system uses the same set of APIs to access common COE components, the same approach to integration, and the same set of tools for enforcing COE principles. Systems are built on top of the COE and use precisely the same COE software components, not just the same algorithms, for common functions (e.g., communications interfaces, dataflow management). This approach to software reuse significantly reduces interoperability problems because if the same software is used, it is not possible to have two systems that interpret or implement standards differently. The next subsection describes the features of GCCS and GCSS in more detail as examples of COE based systems.

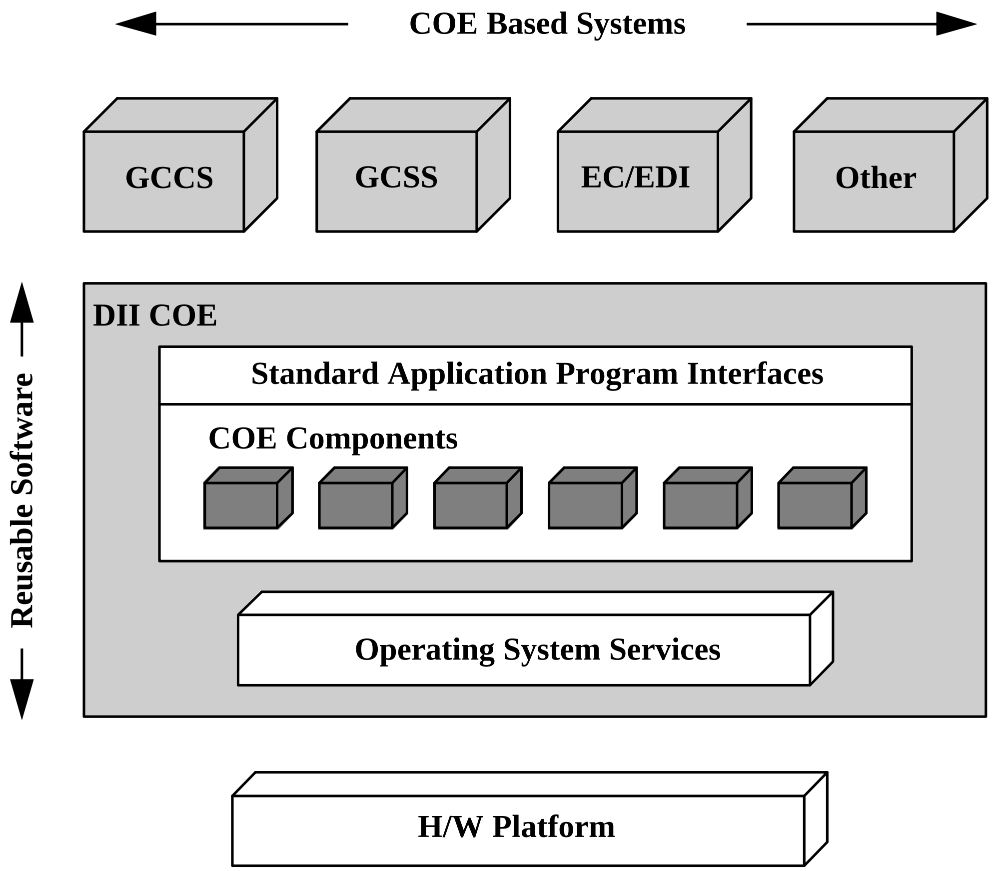

Figure 1-1: DII COE and COE Based Systems

1.1.1.1 GCCS As A COE-Based System

GCCS is a system specifically designed to meet the C4I requirements of the warrior at various echelons within the command structure. It consists of geographically distributed workstations inter-connected via LAN/WAN technologies on a classified network (SIPRNET). The features provided and the LAN/WAN topology allow warriors to collaboratively share mission responsibilities. Collaboration is possible in areas as diverse as creating Time Phased Force and Deployment Data (TPFDD), distributing Air Tasking Orders (ATO), performing intelligence analysis, and maintaining a common view of the battlefield with up-to-date display of the deployment of all joint and enemy forces.

The GCCS system provides a suite of capabilities across a number of mission application areas that include the following:

- Manpower Requirements Analysis
- Force Planning
- Collaborative Mission Planning
- All Source Data Fusion & Correlation
- Office Automation
- Logistics Support
- Status of Readiness Reports
- Cartographic and Imagery Display and Analysis
- Transportation Planning
- Resource Management
- Fuel Resource Planning
- Teleconferencing
- Scheduling and Movement
- Medical Planning
- Comms and Msg Handling
- Intelligence Analysis

The sheer magnitude and capability of GCCS can quickly overwhelm even the most experienced operators. However, the COE provides system administration tools to allow site administrators to selectively install only those software applications required for the site. This minimizes hardware requirements and simplifies site administration. Site administrators can further tailor the installation so that operators are given access to only those applications that pertain to their mission area, or for which they have the proper clearances. GCCS allows an operator to access any function, for which they are authorized, from any workstation so that privileges are tied to the operator, not a specific workstation.

Software updates are periodically made available as new capabilities are developed, or as software patches are created to fix problems. Site administrators can receive these updates via tapes, or electronically across the SIPRNET. Electronic updates are available in either a "push" (e.g., the update process is initiated electronically by a DISA Software Support Activity) or "pull" mode (e.g., the update process is initiated electronically by the operational site).

1.1.1.2 GCSS As A COE-Based System

GCSS is designed to fulfill warfighter acquisition and logistics support functions. As with GCCS, the system consists of geographically distributed workstations inter-connected via LAN/WAN technologies on a classified network. Operators have shared access to technical manuals, drawings, Engineering Change Proposals (ECPs), and status of work in progress regardless of their geographic location. This collaborative feature of GCSS is similar to the teleconferencing capability of GCCS and is supported by the same COE infrastructure. The GCSS system effectively integrates people and organizations, data and information, and work processes across the enterprise.

GCSS provides a comprehensive suite of capabilities related to the acquisition process, and to logistics support. Many of the capabilities, such as office automation, are identical to GCCS and hence use the same COE components. Other requirements, such as the need to support the CALS standard, are GCSS unique. Major features include the following:

- Engineering Drawings Support
- Depot Maintenance Support
- Materiel Management
- Technical Orders
- Workflow Management and Metrics
- Pert Charts
- Logistics Support Analysis
- Access to Non-destructive Imaging Data
- Training Plans
- Reliability Data Management
- Configuration Management
- Teleconferencing
- Office Automation
- CALS Support
- Cost and Schedule Tracking

GCSS uses the same COE system administration tools as GCCS to allow site administrators to selectively install only those software applications required for the site. This minimizes hardware requirements and simplifies site administration. Site administrators can further tailor the installation so that operators are given access to only those applications that pertain to their area of responsibility.

Software updates are periodically made available as new capabilities are developed, or as software patches are created to fix problems. Site administrators can receive these updates via tapes, or electronically in the same manner as GCCS site administrators.

1.1.2 The DII COE As An Architecture

The DII COE is a "plug and play" open architecture designed around a client/server model. Functionality is easily added to or removed from the target system in small manageable units, called *segments*. Segments are defined in terms of functions that are meaninful to operators, not in terms of internal software structure. Structuring the software into segments in this manner is a powerful concept that allows considerable flexibility in configuring the system to meet specific mission needs or to minimize hardware requirements for an operational site. Site personnel perform field updates by replacing affected segments through use of a simple, consistent, graphically oriented user interface.

The DII COE model is analogous to the Microsoft Windows® paradigm. The idea is to provide a standard environment, a set of standard off-the-shelf components, and a set of programming standards that describe how to add new functionality to the environment. The Windows paradigm is one of "federation of systems" in that properly designed applications can coexist and operate in the same environment. But simple coexistence is not enough. It must be possible for applications to share data. While Windows allows some limited data sharing through "cut and paste" between windows, there is no underlying infrastructure for data sharing at a deeper level. The DII COE extends the Windows paradigm to allow for true "integration of systems" in that mission applications share data at the server level.

Federation versus integration is an important architectural advantage. However, integration is not possible without strict standards that describe how to properly build components to add to the system. This document, and other related documents, detail the technical requirements for a well behaved, COE-compliant application. The COE provides automated tools to measure compliance and to pinpoint problem areas. A useful side effect of the tools and procedures is that software integration is largely an automated process, thus significantly reducing development time while automatically detecting potential integration and runtime problem areas.

More precisely, to a developer the DII COE is:

- **An Architecture:** A precisely defined TAFIM-compliant (Technical Architecture Framework for Information Management), client/server architecture for how system components will interact and fit together, and a definition of the system level interface to COE components.

- **A Runtime Environment:** A standard runtime operating environment that includes "look and feel," operating system, and windowing environment standards. Since no single runtime environment is possible in practice, the COE architecture provides facilities for a developer to extend the environment in such a way as to not conflict with other developers.

- **Software:** A clearly defined set of already implemented, reusable functions.

- **APIs:** A collection of Application Programmer Interfaces (APIs) for accessing COE components. Thus, the COE is a set of building blocks in the same sense that X Windows and Motif are building blocks for creating an application's Graphical User Interface (GUI).

DISA maintains the software in an on-line configuration management repository called CSRS (COE Software Repository System). This decreases the development cycle by allowing developers to receive software updates, or to submit new software segments, electronically.

1.1.3 The DII COE As An Implementation Strategy

The COE is also an evolutionary acquisition and implementation strategy. This represents a departure from traditional development programs. It emphasizes incremental development and fielding to reduce the time required to put new functionality into the hands of the warrior, while not sacrificing quality nor incurring unreasonable program risk or cost. This approach is sometimes described as a "build a little - test a little - field a lot" philosophy. It is a process of continually evolving a stable baseline to take advantage of new technologies as they mature and to introduce new capabilities. But the changes are done one step at a time so that the warfighters always have a stable baseline product while changes between successive releases are perceived as slight. Evolutionary development has become a practical necessity for many development programs because the traditional development cycle time is longer than the technical obsolescence cycle time.

From the perspective of a COE-based system, the implementation strategy is to field new releases at frequent intervals. Each release might include enhancements to both the COE and mission area applications. Mission area applications are considered to be provisional, subject to user feedback. Applications for which feedback is favorable are retained in subsequent releases and hardened as needed for continued operational use. As appropriate, mission applications that are widespread in use and commonalty will be integrated into the COE, or evolved to add new features.

The COE implementation strategy is carefully structured to protect functionality contained in legacy systems so that over time they can migrate to full COE utilization. This is achieved through publishing "public" and "private" APIs. Public APIs are those interfaces to the COE that will be supported for the life cycle of the COE. Private APIs are those interfaces that are supported for a short period of time to allow legacy systems to migrate from unsanctioned to sanctioned APIs. All new development is required to use only public APIs and use of any other APIs results in a non-COE compliant segment. The process of migrating from existing legacy "stove-pipe" systems to utilize the COE is a primary source for articulating technical requirements for the COE, and it provides program managers with information useful to establish development priorities.

From the perspective of a system developer, whether developing a new application or migrating an existing one, the COE is an open client/server architecture that offers a collection of services and already built modules for mission applications. Thus, the developer's task is to assemble and customize existing components from the COE while developing only those unique components that are peculiar to particular mission requirements. In many if not most cases, this amounts to adding new "pull down menu entries and icons."

1.2 Lessons Learned

The COE as the embodiment of an architectural concept offers the opportunity to leverage a mature, proven, field tested software base for a wide variety of applications for the services, agencies, and Joint community. As budgets shrink and as budgetary priorities shift, program managers require the ability to continue to respond rapidly with systems that satisfy the information needs of United States and Allied Armed Forces. The COE implementation strategy is a significant advancement in fulfilling this ongoing need.

Examination of state-of-the-art development in light of these realities results in a set of fundamental tenets that greatly influence the history, future, and direction of the DII COE. An explanation of these tenets is useful in understanding the COE as a whole.

¥ *Current practices lead to development and redevelopment of the same functionality across systems.* Redevelopment is frequently necessary because of technological changes as algorithms are improved or as hardware becomes faster and cheaper. However, development cost is often due to a lack of coordination between programs which share common requirements.

¥ *Duplication of functionality within the same system is more expensive than avoiding duplication*. Lack of coordination between program developers is a fundamental cause for duplicative functions, but an additional factor is that reuse libraries are not commonly available. The impact is more than just program costs. System users are often given conflicting information even in the presence of identical data because designers took slightly different approaches to solving the same problems, or made slightly different assumptions.

¥ *Interoperability is not achievable through "paper" standards alone.* Interoperability problems are generally caused, not by the standards chosen, but by differing or incorrect interpretations of standards. System designers often choose different standards with which to comply, but even when the standards are the same, different interpretations of the standards can greatly change the way the resulting system operates. The COE emphasizes use of industry and government standards, but relies even more on automated ways of measuring and evaluating compliance, and thus quantitatively evaluating program risk. The only practical way to achieve interoperability is to use exactly the same software, written to appropriate standards, for common functions across applications. For example, the COE contains a common correlator to ensure that all users see the same tactical picture. The answer produced by the correlator may be incorrect, but it will be incorrect for all users. But this also means that a problem correction in one place then becomes effective for all users.

¥ *Current practices lead to exponential growth in testing and associated development costs.* Lack of commonalty and modularity in system building blocks means that there is much duplication of effort in testing basic functionality and testing in one section of a system is often tightly coupled to testing in another section. This complicates and extends the certification process. Configuration management, system integration, and long term maintenance are also more complex and costly when there is a lack of commonalty and modularity in system building blocks.

¥ *The importance of training is usually underestimated, and the magnitude of the training problem is increasing.* An operator is often expected to use multiple systems which behave completely differently, are equally complex with their own subtleties, and which give slightly different answers. Operator turnover is rapidly reaching the point where the time it takes to train an operator is a significant portion of the time the operator is assigned to his current tour of duty. Training is greatly reduced by a consistent "look and feel" and by the ability to present to the operator only those functions useful for his task.

¥ *Don't reinvent the wheel.* If a component already exists, it should probably be utilized even if the component is not the optimum, best possible solution. Almost any module can be improved but that is rarely the issue. Reuse of existing and proven software allows focus of attention on mission uniqueness. Rather than concentrating scarce development resources on recreating building blocks, the resources can be more appropriately applied to customization and development of functionality that is not already available.

¥ *Utilize existing commercial standards and products whenever feasible.* The commercial marketplace generally moves at a faster pace than the military marketplace and advancements are generally available at a more rapid rate. Use of commercial products has several advantages. Production costs are lowered by using already built items. The probability of product enhancements is increased because the marketplace is larger. The probability of standardization is increased because it is driven by a larger customer base.

1.3 Assumptions and Objectives

The following assumptions apply to the DII COE:

¥ The DII COE will migrate to full compliance with the TAFIM standards profile. These standards promote an open systems architecture, the benefits of which are assumed to be well known and generally accepted.

¥ The DII COE is to be hardware independent and will operate on a range of open systems platforms running under standards-based operating systems. Program driven requirements, associated testing costs, and funding will dictate which specific hardware platforms are given priority.

¥ Non-developmental items (NDIs), including both commercial off-the-shelf (COTS) and government off-the-shelf (GOTS) products, are the preferred implementation approach.

WWMCCS replacement was the main focus for near-term development, while longer-term development is driven by C4I For the Warrior requirements, logistics support requirements for GCSS, and by financial support requirements for EC/EDI. These broad program drivers lead to a number of program objectives that include those stated in the *TAFIM, Volume 2*:

1. **Commonalty**: Develop a common core of software that will form the foundation for Joint systems, initially for C4I and logistics systems.

2. **Reusability**: Develop a common core of software that is highly reusable to leverage the investment already made in software development across the services and agencies.

3. **Standardization**: Reduce program development costs through adherence to industry standards. This includes use of commercially available software components whenever possible.

4. **Engineering Base**: Through standardization and an open architecture, establish a large base of trained software/systems engineers.

5. **Training**: Reduce operator training costs and improve operator productivity through enforcement of a uniform human-machine interface, commonalty of training documentation, and a consistent "look and feel."

6. **Interoperability**: Increase interoperability through common software and consistent system operation.

7. **Scalability**: Through use of the segment concept and the COE architectural infrastructure, improve system scalability so that COE-based systems will operate with the minimum hardware resources required.

8. **Portability**: Increase portability through use of open systems concepts and standards. This also promotes vendor independence for both hardware and software.

9. **Security**: Improve system security.

10. **Testing**: Reduce testing costs because common software can be tested and validated once and then applied to many applications.

1.4 Document Scope

This document describes the technical requirements for building and integrating software components on top of the DII COE. It provides implementation details which describe, from a software development perspective, the following:

¥ the Common Operating Environment (COE) approach to software reuse,
¥ the runtime execution environment,
¥ the requirements for COE compliance,
¥ how to structure components to automate software integration, and
¥ how to electronically submit/retrieve software components to/from the software repository.

This document supersedes all earlier draft versions, presentations, or working group notes. It specifically supersedes all previous JMCIS *COE* and *Integration Standard* documents, and all previous GCCS or DII *Integration Standard* documents. All segments submitted to DISA are required to be in accordance with this document.

1.5 Applicable Documents and Standards

This document is one in a series of related documents which define development requirements, system architecture, engineering tools, and implementation techniques. Many of the documents cited are available on the World Wide Web, or contact the DISA Engineering office for information on how to obtain the desired documents.

Because the COE and COE-based systems are ongoing programs, enhancements and additional features are developed on a regular basis. Documentation updates are regularly released for each of the documents listed here. Be sure to always reference the latest version for the documents listed below, and be aware that many of the documents are being modified and extended to address DII COE-based systems, not just GCCS or GCSS..

DISA, *GCCS Common Operating Environment Baseline*, November 28, 1994. This document is updated for each GCCS release and contains detailed information on the content of each GCCS release. Of particular importance is the exhaustive list of all public APIs included in the release and titles of applicable API documents.

DISA, *GCCS Common Operating Environment Requirements*, August 15, 1994. This document is updated for each GCCS release and contains the requirements definition for the release.

DISA Joint Interoperability Testing Command (JITC), *GCCS 3.0 Compliance Program Plan*, Draft Version. This document is a proposal for how to formally test segments for GCCS COE compliance. It is currently in draft form and is undergoing revision. It will form the basis for measuring COE compliance across all COE-based systems.

DoD, *Technical Architecture Framework for Information Management*. This is a multi-volume document which defines a standards profile and the DoD Technical Reference Model (TRM) for information management systems.

Institute for Defense Analyses (IDA), *Architectural Design Document for the Global Command and Control System (GCCS) Common Operating Environment (COE)*. This document is the definitive technical description of the COE. It documents the architectural design produced by the GCCS COE Design Working Group. This document will be extended to encompass the DII COE for all COE-based systems.

NRaD, *User Interface Specification for GCCS*. This document, sometimes called the *Style Guide*, defines the "look and feel" for developing user interfaces for GCCS. This style guide is closely patterned after the commercial Motif style guide. This guide is presently being extended to address all COE-based systems, and is being extended to include Microsoft Windows and Windows NT styles.

The following documents are useful for a historical perspective on the evolution of COE concepts. The list is not comprehensive, but gives previous versions of documents which preceded this one.

INRI, *Joint Maritime Command Information System (JMCIS) Common Operating Environment (COE)*, Version 1.3. This document defines the runtime environment requirements for JMCIS segments.

NRaD, *Joint Maritime Command Information System (JMCIS) Integration Standard*, Version 2.0. This document defines requirements for how developers are to submit software segments to the SPAWAR Software Support Activity (SSA) at the Navy Command, Control, and Ocean Surveillance Center Research, Development, Test & Evaluation Division (NRaD) in San Diego, California. It also contains information on the Navy's JMCIS On-line Library (JOL) which served as a foundation for creating CSRS.

NRaD, *Global Command and Control System Integration Standard*, Version 1.0. This document contains information combined from the two referenced JMCIS documents as adapted for use in GCCS.

1.6 Document Structure

This document is structured to correspond to the typical phases in a development cycle, beginning with how a developer builds a segment, submits it to the government, and then how it is fielded to an operational site. Chapter 1 of this document is an overview of the DII COE, a brief history of its development, and applicable documents and standards.

Chapter 2 gives a brief technical description of the COE, it's components, and the principles which determine whether a software component is part of the COE or is a mission application. Selection of the particular components contained in the COE determine what applications can be supported, but the principles which define a COE are not application specific.

Chapter 3 is an overview of the development process. It includes a discussion of the process from segment registration through development, submission to CSRS, integration, and site installation. The tools provided in the COE and how they are used is key to understanding automated integration.

Chapter 4 describes database considerations within the context of the COE. Databases are heavily used within COE-based systems, and early consideration of their structure, how they are to be used, and how they are to fit into the overall system is crucial in building a successful system.

Chapter 5 describes the runtime environment as it exists for operational sites, the disk directory and file structure fundamental to the COE, and the procedures for integrating segments into a runtime environment. Requirements detailed in Chapter 5 must be carefully followed so that applications will not interfere with each other, and so that integration is largely an automated process.

Chapter 6 provides some suggestions for setting up a software development environment. Few requirements are stipulated for a development environment to allow as much freedom for developers and program managers as possible.

Chapter 7 describes two important components for both developers and operational sites: the on-line COE Software Repository System (CSRS), and the COE Information Server (CINFO). These components are used to disseminate and manage software, documentation, meeting notices, and general information of importance to the COE community.

Appendix A lists the currently supported COE configurations. The appendix includes supported hardware, and supported COTS versions.

Appendix B presents a checklist for developers to use as an aid in determining the degree to which a segment is COE compliant. As described in the appendix, some conditions are mandatory, others require a migration strategy to show conformance, while others are optional but recommended.

Appendix C describes the automated tools provided with the COE. The philosophy is to provide developers with access to the same tools that integrators will use so that as much as possible, segment integration is performed by the segment developers prior to segment delivery. Integration of segments with the COE is the responsibility of the segment developer. Government integrators serve as validators *only* in this process to ensure that developers produce COE-compliant segments. In addition to segment validation, government integrators perform system level integration of all segments submitted by all developers to create the target system.

Appendix D gives additional information on the on-line repository CSRS, and the information server CINFO.

Segment registration is required so as to identify potential conflicts as early in the development cycle as possible. Appendix E describes how to register a segment, and what information is required for registration.

Appendix F provides additional database related information. It identifies RDBMS vendor specific considerations.

Appendix G is a draft supplement that describes how to build PC-based segments. It also describes the purpose and operational use of PCs in the context of COE-based systems.

The Glossary contains a definition of commonly encountered terms.